

TP1 MPSI ITC 2025

Rappels généraux

- Écrire chaque fonction **sans** utiliser **return** ou **break** à l'intérieur des boucles.
- Soigner les noms de variables et les tests.
- Pour la variance, utiliser la formule :

$$\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

1) Conditions

1.1 Deux fonctions avec `if ... else`

Question 1.

📌 maximum de deux entiers

```
1 def max2(a: int, b: int) -> int:
2     """Retourner le maximum de a et b en utilisant if ... else."""
3     # A ECRIRE
4     pass
```

```
>>> max2(3, 7)
7
>>> max2(10, -2)
10
```

Question 2.

📌 pair ou impair

```
1 def pair_ou_impair(n: int) -> str:
2     """Retourner 'pair' si n est pair, sinon 'impair'."""
3     # A ECRIRE
4     pass
```

```
>>> pair_ou_impair(12)
'pair'
>>> pair_ou_impair(7)
'impair'
```

1.2 Deux fonctions avec `if ... elif ... else`

Question 3.

📌 signe d'un nombre

```

1 def signe(x: float) -> int:
2     """Retourner -1, 0 ou 1 selon le signe de x (if/elif/else)."""
3     # A ECRIRE
4     pass

```

```

>>> signe(-3.5)
-1
>>> signe(0.0)
0
>>> signe(2.7)
1

```

Question 4.

📌 classification d'une note

```

1 def appreciation(note: float) -> str:
2     """Classifier une note sur 20 avec if/elif/else."""
3     # A ECRIRE
4     pass

```

```

>>> appreciation(8.5)
'AJ'
>>> appreciation(12.0)
'Passable'
>>> appreciation(14.5)
'Bien'
>>> appreciation(18.0)
'Très bien'

```

2) Boucle for : min, max, moyenne, variance

Question 5.

📌 minimum d'une liste

```

1 def min_liste(xs: list[float]) -> float:
2     """Retourner le minimum d'une liste non vide en utilisant for."""
3     # A ECRIRE
4     pass

```

```

>>> min_liste([3, -2, 7, 0])
-2

```

Question 6.

📌 maximum d'une liste

```

1 def max_liste(xs: list[float]) -> float:
2     """Retourner le maximum d'une liste non vide en utilisant for."""
3     # A ECRIRE
4     pass

```

```
>>> max_liste([3, -2, 7, 0])
7
```

Question 7.

📌 moyenne d'une liste

```
1 def moyenne(xs: list[float]) -> float:
2     """Calculer la moyenne d'une liste non vide (boucle for)."""
3     # A ECRIRE
4     pass
```

```
>>> moyenne([10, 12, 14])
12.0
```

Question 8.

📌 variance d'une liste

```
1 def variance(xs: list[float]) -> float:
2     """Calculer Var = E[X^2] - (E[X])**2 en utilisant for."""
3     # A ECRIRE
4     pass
```

```
>>> variance([1, 2, 3, 4])
1.25
```

3) Boucle while : recherches et parcours

Question 9.

📌 présence d'un élément

```
1 def contient(xs: list[int], x: int) -> bool:
2     """Tester la présence de x dans xs avec une boucle while."""
3     # A ECRIRE
4     pass
```

```
>>> contient([4, 1, 9, 1], 9)
True
>>> contient([4, 1, 9, 1], 7)
False
```

Question 10.

📌 indice de la première occurrence

```
1 def indice_premier(xs: list[int], x: int) -> int:
2     """Renvoyer l'indice de la première occurrence de x avec while, sinon -1."""
3     # A ECRIRE
4     pass
```

```
>>> indice_premier([5, 8, 5, 2], 5)
0
>>> indice_premier([5, 8, 5, 2], 7)
-1
```

Question 11.

📌 indice de la dernière occurrence

```
1 def indice_dernier(xs: list[int], x: int) -> int:
2     """Renvoyer l'indice de la dernière occurrence de x avec while, sinon -1."""
3     # A ECRIRE
4     pass
```

```
>>> indice_dernier([5, 8, 5, 2], 5)
2
>>> indice_dernier([1, 2, 3], 9)
-1
```

Question 12.

📌 compter les occurrences

```
1 def compter(xs: list[int], x: int) -> int:
2     """Compter le nombre d'occurrences de x dans xs avec while."""
3     # A ECRIRE
4     pass
```

```
>>> compter([2, 2, 2, 1], 2)
3
```

Question 13.

📌 liste triée croissante ?

```
1 def est_triee(xs: list[int]) -> bool:
2     """Retourner True si xs est croissante, False sinon (while)."""
3     # A ECRIRE
4     pass
```

```
>>> est_triee([1, 2, 2, 4])
True
>>> est_triee([3, 1, 4])
False
```

4) Plus difficile

Question 14.

📌 PGCD (algorithme d'Euclide)

```

1 def pgcd(a: int, b: int) -> int:
2     """Calculer le PGCD d'a et b (a>=0, b>=0) avec l'algorithme d'Euclide."""
3     # A ECRIRE
4     pass

```

Principe : à chaque tour, l'ancien diviseur devient le nouveau dividende et l'ancien reste devient le nouveau diviseur. On renvoie l'avant dernier reste non nul.

```

>>> pgcd(54, 24)
6
>>> pgcd(21, 14)
7

```

Question 15.

📌 fusion de deux listes triées

```

1 def fusion_tries(a: list[int], b: list[int]) -> list[int]:
2     """
3     Fusionner deux listes triées a et b en une nouvelle liste triée.
4     Utiliser des indices i, j et une boucle while.
5     """
6     # A ECRIRE
7     pass

```

```

>>> fusion_tries([1, 3, 5], [2, 4, 6, 6])
[1, 2, 3, 4, 5, 6, 6]
>>> fusion_tries([], [0, 1])
[0, 1]

```

Question 16.

📌 Inverser une liste sans effet de bord

```

1 from typing import List, TypeVar
2 T = TypeVar("T") #pour définir une décoration de type générique
3
4 # inverser sans effet de bord
5 def inverser(xs: List[T]) -> List[T]:
6     """Retourner une nouvelle liste qui est xs renversée (pas d'effet de bord).
7     [1,2,3] -> [3,2,1]
8     """

```

Remarque. « Pas d'effet de bord » signifie que la liste passée en paramètre ne doit pas être modifiée.

Question 17.

📌 Inverser une liste AVEC effet de bord

```

1 # inverser sans effet de bord
2 def inverser_en_place(xs: List[T]) -> None:
3     """Renverser xs en place (effet de bord)."""

```

Question 18.

📌 Rotation sans effet de bord (pas de slicing)

```
1 # rotation sans effet de bord (pas de slicing)
2 def rotation(xs: List[T], k: int) -> List[T]:
3     """
4     Rotation à gauche de k éléments, sans effet de bord et sans slicing.
5     Ex.: rotation([1,2,3,4,5,6], 2) -> [3,4,5,6,1,2]
6     """
```

Question 19.

📌 Rotation avec effet de bord (pas de slicing)

```
1 # rotation_en_place avec effet de bord
2 def rotation_en_place(xs: List[T], k: int) -> None:
3     """
4     Rotation à gauche de k éléments, en place (effet de bord),
5     Ex.: xs = [1,2,3,4,5,6]; rotation_en_place(xs, 2) -> xs devient [3,4,5,6,1,2]
6     """
```